HAL open science

Working Memory for Assessment Under Inconsistency

Pierre Bisquert, Florence Dupin de Saint-Cyr

To cite this version:

Pierre Bisquert, Florence Dupin de Saint-Cyr. Working Memory for Assessment Under Inconsistency. NMR 2021 - 19th International Workshop on Non-Monotonic Reasoning, Nov 2021, Hanoï, Vietnam. pp.149-158. hal-03327998

HAL Id: hal-03327998

https://hal.science/hal-03327998v1

Submitted on 27 Aug 2021

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Working Memory for Assessment Under Inconsistency*

Pierre Bisquert[1,2] Florence Dupin de Saint-Cyr[3]

August 2021

[1]IATE, Univ Montpellier, INRAE, Institut Agro, Montpellier, France
[2]LIRMM, Inria, Univ Montpellier, CNRS, Montpellier, France
[3]IRIT, Toulouse University
pierre.bisquert@inrae.fr florence.bannay@irit.fr

Abstract

This paper proposes a new way of handling the inconsistency of a knowledge base when answering to a query about the validity of a formula. The idea is inspired by human behavior in front of inconsistency, namely, try to never encounter it. For this purpose, we encode a kind of compartmentalization of the working memory. More precisely, given a query and a potentially inconsistent knowledge base, called long term memory, our system only loads in working memory the consistent knowledge which is the most related to the query. We position this system with regard to a major reference in the field, Brewka's preferred subtheories, and study its efficiency by providing complexity and experimental results.

keywords: Inconsistency handling, SAT, Bounded rationality, Preferred subtheories

1 Introduction

How do humans reason in the presence of contradictory information? Some psychologists [17] answer that they inhibit counter-examples to come to their mind. Translating this phenomena inside a framework that uses the distinction done by [4] between long-term (*LTM*) and working memory (*WM*) where *WM* is conceived as the "activated" part of the long-term memory (*LTM*), [5] studies how memory activation is produced or inhibited. Findings of [18] corroborate that *WM*-resources are used for retrieval and inhibition of stored counter-examples and for avoiding conflicts with the logical validity of a reasoning problem [18]. Inspired by the way inconsistency seems to be handled by human beings we propose a model that tries to reason with two datasets: a first potentially inconsistent one representing the *LTM* and a second consistent one for representing the *WM*.

Handling of inconsistent knowledge bases is a thoroughly studied subject in computer science, and in particular in the domains of Databases (with Repairs [24], Consistent Query Answering [15]) and Knowledge Representation and Reasoning (SAT, repairs, revision, argumentation), see chapters [1, 20]. In general, such an inconsistency is addressed by either repairing directly the knowledge base, which typically lead to loss of information [19, 8]), or by considering that the query is entailed if it is in the intersection of all the minimal repairs (or maximal maxi-consistent subbases). This last approach, while avoiding the loss of information, is hampered by the computational price required to compute all the repairs or subbases. Another family of approaches for inconsistency handling, called paraconsistent logic, is outside the scope of this paper since these approaches either rely on extra-information (e.g. possibilistic logic [21]) or are based on non-classical logical axioms (see the surveys [3, 12]).

*This is a draft version the paper has been accepted in NMR 2021: 19th International Workshop on Non-Monotonic Reasoning November 3-5, 2021, Hanoi, Vietnam (VIRTUALLY)

In this paper, we use ideas from psychology as an inspiration to provide a system able to handle inconsistent knowledge base without needing to compute all the maximally consistent subbases. Indeed, the notion of *WM* allows for the computation of one "repair" in the sense that only relevant and consistent pieces of information from the knowledge base are activated, effectively compartmentalizing the inconsistency. The paper will present a way of recursively selecting which pieces should be activated based on some heuristic, and how it will affect the reasoning, in complexity and in execution time, notably when the *WM* is restricted in size to account for bounded rationality [37].

We are in particular interested in the impact of successive querying in this kind of context where results might differ due to the way the size-limited subbase is built. In order to experiment our approach, we will place ourselves in the context of SAT. Indeed, SAT is a deeply studied domain with powerful solvers (see the competition [25]). On that topic, it should be noted that within the SAT domain, handling inconsistency can be done thanks to (weighted) MAXSAT [30]. While this is a relevant approach, it needs either the removal cost of each clause, which needs to be elicited, or to consider that every clause is as relevant to the query as any other. In this work, we would like to study another, somehow close, way where the selection of clauses is based on their closeness to a given query.

The paper is organized as follows: we start by recalling inconsistency handling classical approaches, then we describe the Working Memory approach. In the last section we demonstrate theoretical results about this approach and study its complexity, then we describe the experiments that were conducted on several benchmarks. We conclude by a discussion about MAXSAT and about the non-monotonic properties of the inference relation based on the Working Memory approach.

2 Background on selection-based inconsistency handling

Notations: we consider a propositional logical language $\mathscr{L}$ containing formulas denoted by lower case Greek letters, based on a vocabulary $\mathscr{V}$ of variables denoted by Latin lower case letters. Negation, conjunction, disjunction, material implication, contradiction and classical inference are denoted respectively by $\neg$, $\wedge$, $\vee$, $\rightarrow$, $\perp$ and $\vdash$. A CNF formula is a conjunction of clauses, a clause is a disjunction of literals, a literal is a variable or its negation. Abusing notation, clauses are assimilated to sets of variables when using the two set operators $\in$ and $\cap$ (for membership and intersection). Lists of elements are represented with square brackets, and :: is the operator s.t. $e{::}L$ is the new list formed by the element e followed by the elements of L.

In this paper we are going to use finite knowledge bases defined as follows:

Definition 1 (Knowledge base)**.** *A knowledge base is a finite set of formulas of $\mathscr{L}$, considered as the conjunction of its elements.*

One of the best known approach to cope with inconsistency is the one of Rescher and Manor [36]: it is based on the computation of the set of maximal consistent subsets of the belief base, then a formula is accepted as a consequence of the base if it can be classically inferred from every maximal consistent subset (or MSS for maximum satisfiable subset). This idea has been refined in the preferred subtheory approach of [10] where the knowledge base is divided into several subsets according to a given reliability level. We will see in Section 3.2 that our approach actually allows to dynamically compute this reliability levels with respect to a given query.

Definition 2 (Preferred subtheory [10])**.** *Given a tuple $T = (T_1, \ldots, T_N)$ of sets of formulas*[1] *of $\mathscr{L}$, $S = S_1 \cup \cdots \cup S_N$ is a* preferred subtheory *of T* iff *for all k, $(1 < k < N)$ $S_1 \cup \cdots \cup S_k$ is a maximal consistent subset of $T_1 \cup \cdots \cup T_k$.*

[1] Preferred subtheories are originally defined of a first-order language; in this paper, we will restrict this defining by using a propositional language.

In the words of Brewka: "in order to obtain a preferred subtheory of T we have to start with any maximal consistent subset of T_1, add as many formulas from T_2 as consistently can be added (in any possible way), and continue this process for $T_3, \ldots, T_N$".

Computing preferred subtheories requires some kind of inconsistency checking; in this paper, we will make use of the notion of minimal unsatisfiable subsets[2].

Definition 3 (MUS and MSS [31])**.** *A subset S of clauses of a base B is a minimal unsatisfiable subset (MUS) if S is inconsistent and for all $c \in S$, $S \setminus \{c\}$ is consistent.*

A subset $S \subseteq B$ is a maximal satisfiable subset MSS if S is consistent and for all $c \in B \setminus S$, $S \cup c$ is inconsistent.

Example 1. *Let us consider the following knowledge base LTM (in CNF form):*

$$LTM = \overbrace{(a \vee \neg d)}^{c_1} \wedge \overbrace{\neg a}^{c_2} \wedge \overbrace{(\neg a \vee b \vee d)}^{c_3} \wedge \overbrace{\neg b}^{c_4} \wedge \overbrace{(\neg a \vee c)}^{c_5} \wedge \overbrace{\neg c}^{c_6} \wedge \overbrace{d}^{c_7}$$

There are two MUSes of LTM: $\{c_1, c_2, c_7\}$ and $\{c_1, c_5, c_6, c_7\}$. There are four MSSes of LTM: $\{c_1, c_2, c_3, c_4, c_5, c_6\}$, $\{c_1, c_3, c_4, c_5, c_7\}$, $\{c_1, c_3, c_4, c_6, c_7\}$ and $\{c_2, c_3, c_4, c_5, c_6, c_7\}$.

When the user has information about the formulas that are more important/sure (called "preferred" in [10]) then the selection can be done among the preferred subtheories (which according to [10] are maximal consistent subbases of the knowledge base in which the most important formulas are primarily chosen). Nebel in [33] also proposes to use a syntax based approach that he calls "epistemic relevance", which is a complete preorder on all the formulas that are consequences of the beliefs. This relevance/preference information may come from the confidence given into the different sources of the belief base, it is then considered as exogeneous extra information about the beliefs. Another kind of approach takes profit from the syntax of the belief base to discover the strength of each belief, this meta information is then endogeneous with respect to the belief base: for instance System Z is able to rank automatically the beliefs based on their specificity [35].

When a user wants to conserve the belief base without forcing consistency, two ways can be adopted: either reason on ALL the most interesting subbases (for instance, reason on Brewska's preferred subtheories [10], providing that rankings on beliefs are available, or simply maximal consistent subbases when no extra-information is available) or select only ONE preferred consistent subbase and reason with it [7, 6]. In both cases, the whole initial base is preserved, but the reasoning process is made on one (or several) of its consistent part(s) (called "repair(s)" in Query Answering community [24, 15]).

Other, somewhat different, approaches for reasoning under inconsistency exist; for instance, modifying the beliefs by adding premises in order to specify better the context in which some rules should not be fired [19, 22], somehow coming back to the old idea of circumscription [32].

3 Working Memory

In this section we are going to present a way to assess a CNF formula called the query, denoted φ, w.r.t. a potentially inconsistent finite CNF knowledge base called *LTM* (for long term memory). This is done by checking the consistency of the formula w.r.t. a subset of the *LTM* called *WM* (for working memory).

More precisely, we propose a process that: 1) links the *LTM* clauses together based on the number of common literals (Section 3.1), 2) uses these links to build a *WM* that is relevant for the query φ to assess, where relevance is understood in terms of common variables (Section 3.2) and 3) actually checks the status of the query (or queries, Section 3.3).

[2]Please note that our approach is agnostic on that point and might use different inconsistency checking mechanisms. That being said, while computing MUS is computationally expensive, they need to be computed just once, which would not potentially be the case with other inconsistency checking methods where the computation would be needed with each newly considered clauses.

Algorithm 1: LTM_preprocessing(LTM)

Input: LTM in Dimacs CNF format
Output: Var2Cl: dict. of clauses associated with var; maxCom: max nb of common vars in 2 clauses; AssocCl: dictionary of associated clauses

```
Var2Cl ← empty dictionary
for each clause c in LTM do for each v in c do Var2Cl(v) ←Var2Cl(v) ∪ {c}
maxCom ← 0; AssocCL ← empty dictionary
for each (c1,c2) in LTM^2 s.t. c1 ∩ c2 ≥ 1 do
    AssocCl(c1) ← (c2,c1 ∩ c2) :: AssocCl(c1)
    if maxCom < c1 ∩ c2 then
        maxCom ← c1 ∩ c2
return (Var2Cl, maxCom, AssocCl)
```

Algorithm 2: Query_preprocessing(φ,LTM)

Input: φ: a query in CNF format; LTM: set of formulas in CNF format
Output: QAssocV: dict. of common var of each LTM clause w.r.t. φ

```
QAssocV ← empty dictionary of clauses
for each clause c in LTM do
    for each clause i in φ do
        comV ← i ∩ c                                  // common vars between i and c
        if comV ≠ ∅ then
            QAssocV(c) ← QAssocV(c) ∪ comV
return (QAssocV)
```

3.1 Preprocessing on LTM

This first step of preprocessing on LTM extracts the information required to build a consistent and relevant *WM*. More precisely, it consists in creating a dictionary *AssocCl* that associates to each clause c the list of clauses that contains at least one common variable with c together with the number of common variables:

$$AssocCl(c) = [(c', nbV) \quad | \quad c' \in LTM \setminus \{c\}, \; nbV = |c \cap c'| \text{ s.t. } nbV > 0]$$

This is done by Algorithm 1 which also computes the maximum number of common variables *maxCom* between any pair of clauses and the dictionary $Var2Cl$ which maps each variable v to the set $Var2Cl(v)$ of clauses in which it appears:

$$Var2Cl(v) = \{c \quad | \quad c \in LTM \text{ s.t. } v \in c\}$$

Please note that these association tables do not need to be recomputed for each query (but they can be updated).

Example 2. *In Example 1,* $Var2Cl(a) = \{c_1, c_2, c_3, c_5\}$ *and table AssocCl is:* $AssocCl(c_1) = [(c_2, 1), (c_3, 2), (c_5, 1), (c_7, 1)]$, $AssocCl(c_2) = [c_1, 1), (c_3, 1), (c_5, 1)]$, $AssocCl(c_3) = [(c_1, 2), (c_2, 1), (c_4, 1), (c_5, 1), (c_7, 1)]$, $AssocCl(c_4) = [(c_3, 1)]$, $AssocCl(c_5) = [(c_1, 1), (c_2, 1), (c_3, 1), (c_6, 1)]$, $AssocCl(c_6) = [(c_5, 1)]$, $AssocCl(c_7) = [(c_1, 1), (c_3, 1)]$ *and* $maxCom = 2$.

3.2 Building a consistent *WM* for assessing φ

Given a query φ and a LTM, Algorithm 2 builds the dictionary $QAssocV$ that associates to each clause c of the LTM the set $QAssocV(c)$ of variables that c has in common with φ:

$$QAssocV(c) = c \cap \varphi$$

Algorithm 3: WM_download

Input: QAssocV: dict. of vars assoc. with LTM clauses; AssocCl: dict. of associated clauses in LTM; MUS: set of MUS of LTM; m: capacity size of WM; maxCom: max common vars between 2 clauses

Output: new WM

```
/* Create a max binary heap with clauses associated with their score w.r.t. φ */
Queue ← empty maximal binary heap
maxScore ← 0
for each key c in QAssocV do
    score(c) ← |QAssocV(c)|
    if maxScore < score(c) then maxScore ← score(c)
for each key c in QAssocV do  Add(Queue,(c,score(c)/maxScore))
WM ← empty set; InconsSeen ← empty set
/* Best first search algorithm */
while Queue not empty and |WM| < m do
    (key,rel) ← Remove(Queue)                    // Removing the max element of the heap
    if key ∉ WM then
        if ∃mus ∈ MUS, mus ⊆ WM ∪{key} then // key is inconsistent with WM
            InconsSeen ← InconsSeen ∪ {key}
        else // key is consistent with WM
            WM ← WM ∪{key}
            keyAdjacents ← AssocCl(key)
            for each pair (c, s) in keyAdjacents do
                if c ∉ WM and c ∉ InconsSeen then
                    Add(Queue, (c,rel × s/maxCom))
return (WM)
```

This dictionary will be used (in Algorithm 3) to start feeding the WM with clauses directly linked to the query, since as detailed below, the score of a clause is the sum of the number of common variables, this number is normalized by dividing it by the maximum number obtained for an entry of the dictionary.

Example 3. *Let us consider the following formula containing two clauses:* $\varphi = (b \vee \neg c) \wedge d$. $QAssocV(c_1) = QAssocV(c_7) = \{d\}$ *meaning that* c_1 *(and also* c_7*) has the variable* d *in common with* φ. $QAssocV(c_3) = \{b, d\}$, $QAssocV(c_4) = \{b\}$, $QAssocV(c_5) = QAssocV(c_6) = \{c\}$.

We propose a best-first-search algorithm (Algorithm 3) for filling the Working Memory with the clauses related to a query φ. Technically, the idea is to select the "closest" clauses w.r.t. to φ, with the closeness notion defined inductively as follows: first the clauses that have a maximum number of common variables with φ are inserted in a maximal binary heap[3] with a *percentage of relevance to the query* (the number of common variables normalized with the maximal score $maxScore$, the biggest number of associated clauses). A clause c with highest relevance is then taken out of the heap and added to the WM (if not inconsistent); its closest clauses (according to AssocCl) c' are added to the heap with a *percentage of relevance to the query* equal to the relevance of c multiplied by its degree of relevance with c' (number of common variables with c' divided by $maxCom$ the maximum of common variables between two clauses in LTM). Using percentage and normalized score ensures that the "farther" from the query a clause is, the lower its relevance to the query will be.

More fundamentally, Algorithm 3 presents a way to compute a relevance score which plays a similar role as Brewka's reliability level, with the significant difference of being dynamically computed. Indeed, first, the set of clauses with the highest relevance to the query, i.e. with the highest number of common variables, is selected to form the first stratum of the knowledge base LTM_1; from this stratum is extracted a set of maximally consistent clauses WM_1. A new relevance score is then computed for the remaining clauses of the LTM based on their relevance with the

[3] A maximal binary heap is represented by a tabular where the root is at index 1, the left child of any node i is at index $2i$ and the right child is at index $2i + 1$.

clauses in WM_1, essentially computing a transitive relevance to the query, and forming LTM_2. This process continues recursively until no new stratum can be formed, either because the maximal size of the WM has been reached or because there is no relevant clause anymore.

More formally, the selection of the relevant clauses requires the notion of consistent sets of clauses that are maximal for inclusion with the condition that they have a size under a given bound s, defined as follows:

Definition 4 (Max-consistent for inclusion under s)**.** *S is a max-consistent subset of K for inclusion under $s \in \mathbb{N}$ (S mci_s K) iff $S \subseteq K$ and S is consistent and $|S| \leq s$ and there is no S' consistent s.t. $S' \subseteq K$ and $|S'| \leq s$, $S' \supset S$.*

Algorithm 3 recursively collects clauses that are less and less (transitively) relevant to the query, yielding a consistent set WM, more formally defined as follows:

Definition 5 (Working memory w.r.t. a query)**.** *Given a knowledge base LTM and a formula φ (called the query), a working memory $WM(LTM, \varphi, m)$ associated with LTM and φ given a maximum size m of the working memory is recursively defined as follows:*

$$score_1(c) = |\bigcup_{c' \in \varphi}(c' \cap c)|, \qquad c \in LTM$$

$$LTM_1 = \operatorname{argmax}_{\substack{c \in LTM \\ \text{and } score_1(c) \neq 0}} score_1(c)$$

$$WM_1 \quad mci_m \quad LTM_1$$

Given $WM_1, \ldots, WM_k$, and $LTM_1, \ldots, LTM_k$ and $m(k) = m - \sum_{i=1}^{k} |W_i|$:

- *If $m(k) > 0$ and $WM_k \neq \emptyset$ then*

$$score_{k+1}(c) = \max_{c' \in WM_k}(|c' \cap c| \times score_k(c')/maxCom)$$

$$\text{where } maxCom = max_{c,c' \in LTM}|c \cap c'|.$$

$$LTM_{k+1} = \operatorname{argmax}_{\substack{c \in LTM \setminus (LTM_1 \cdots LTM_k) \\ \text{and } score_{k+1}(c) \neq 0}} score_{k+1}(c)$$

$$WM_{k+1} \quad mci_{m(k)} \quad LTM_{k+1}$$

- *Else $WM(LTM, \varphi, m) = \bigcup_{j=1}^{k} WM_j$*

As it can be seen in Definitions 4 and 5 with the notion of max-consistent subset for inclusion under s (mci_s), consistency must be maintained when building the WM. In Algorithm 3, we use MUS to ensure consistency, more precisely, each time a new clause is added to the WM we check whether a MUS is not a subset of the WM. The MUS of the LTM are precomputed offline thanks to a standard algorithm (we have chosen to use the system CAMUS [31]). Note that another technique would be to use an incremental SAT solver like GlucoseInc [2], or even just check for consistency each time a new clause is added to the WM. The comparison, in terms of complexity and computation time, of these different consistency handling techniques is left for future work.

Example 4. *Let us consider that we want to build a WM of size $m = 5$ extracted from the LTM of Example 1 for the query $\varphi = (b \vee \neg c) \wedge d$. The scores of the clauses indexed in QAssocV are: $score(c_1) = score(c_4) = score(c_5) = score(c_6) = score(c_7) = 1$ and $score(c3) = 2$ (maxScore), yielding to an initial Queue consisting in the maximal binary heap $[(c_3, 1); (c_1, 0.5); (c_4, 0.5); (c_5, 0.5); (c_6, 0.5); (c_7, 0.5)]$. Algorithm 3 starts building a WM by removing the maximal element*

(the clause that is the most related to the query) of the heap, namely c_3, and adding it to WM. After this step, WM $= \{c_3\}$ and Queue $= [(c_7, 0.5); (c_1, 0.5); (c_4, 0.5); (c_5, 0.5); (c_6, 0.5)]$[4].

The clauses associated to c_3 are already stored in $AssocCl(c_3) = \{(c_1, 2), (c_2, 1), (c_4, 1), (c_5, 1), (c_7, 1)\}$, each of them is added to the Queue with a weight equal to the value of c_3 (which equals 1) times their weight divided by maxCom (which equals 2), yielding the new Queue: $[(c_1, 1); (c_1, 0.5); (c_7, 0.5); (c_5, 0.5); (c_6, 0.5); (c_4, 0.5); (c_2, 0.5); (c_4, 0.5); (c_5, 0.5); (c_7, 0.5)]$. Note that the queue may contain several occurrences of the same element. Then c_1 is removed from Queue and added to the WM, afterwards c_4 then c_7 and c_5. Finally when c_2 is at the top of the heap, it cannot be added since inconsistent with WM idem for c_6. At the end WM $= \{c_1, c_3, c_4, c_5, c_7\}$ (which is actually a max consistent subset of B, it is not necessarily the case that a whole MSS is obtained). Note that due to equalities other WM are obtainable (more precisely, every subset of size 5 of any MSS except the subsets of $\{c_2, c_3, c_4, c_5, c_6, c_7\}$ (since c_1 should be present due to its high number of common variables with the query).

3.3 *WM* loading with overflow for new queries

Once a *WM* has been built, it is possible to evaluate the query. In particular, we will say that a query φ is accepted when $WM \cup \{\varphi\} \not\models \bot$. Note that we used Sat4J [29] to check satisfiability, but any other SAT solver could be used.

Example 5. *Given WM $=\{c_1, c_3, c_4, c_5, c_7\}$ with $\varphi = (b \vee \neg c) \wedge d$, we get $WM \cup \{\varphi\} \models \bot$.*

When a new query φ' arrives, the previously loaded clauses might be irrelevant, i.e. there might be no association between the query and clauses in the *WM* ($AssocCl \cap QAssocV = \emptyset$), which prompts for the loading of other clauses. Then two cases might arise according to the room left in the current *WM* (where $room = m - |WM|$, i.e. capacity size of the *WM* minus current occupation) and to the number of clauses needed to answer query φ' ($needed = |WM(LTM, \varphi', m)|$):

- either the *WM* still has room to store the newly needed clauses: $needed \leq room$, in that case the process is the same as before,
- or the *WM* lacks room: $needed > room$ then a set of old clauses (of size $needed - room$) is discarded from the *WM*.

4 Characterization about efficiency in time and accuracy

In this section, we will study some properties of the inference relation induced by our framework and assess it experimentally.

4.1 Theoretical results

Before getting into the details of the inference relation, we need to define the notion of cluster.

Definition 6 (Clusters)**.** *Given a set of clauses LTM, a* cluster *of the LTM is a set of clauses composing a connected component of the graph whose vertices are the clauses and the edges are relating two clauses with at least one common variable; clusters(LTM) is the set of clusters of LTM.*

Now, we are in the position to define the inference relation with regards to the *WM*.

Definition 7 (*LTM* inference)**.** *Given a LTM and a capacity size m of the WM,*

$$\alpha \mid\!\sim^{m}_{LTM} \beta \text{ is defined by } WM(LTM, \alpha, m) \vdash \alpha \rightarrow \beta$$

[4]Removing the max element of a heap replaces it with the last leave of the tree and percolate it down to its right place, here c_7 go to the top and stays there.

where $WM(LTM, \alpha, m)$ is a working memory in the sense of Definition 5 and $\vdash$ is classical logic inference.

The following proposition establishes that detecting inconsistency of the LTM with the query φ by selecting a consistent subbase with no limit of size is the same as doing it with a size equal to the size of the maximal cluster of the LTM. The proposition holds when the query φ is related to only one cluster of the LTM, in other words when φ concerns only one domain of knowledge (i.e., associated to only one vocabulary).

Proposition 1. *Let $mc = \max_{C \in clusters(LTM)} |C|$, if $mc \leq m$ (where m is the capacity of the WM), and $\alpha \in \mathscr{L}$ s.t. there is only one cluster $C \in clusters(LTM)$ where for all clause i in α, for all cluster C' in $clusters(LTM) \setminus C$ and for all clause $c \in C'$, $i \cap c = \emptyset$:*

$$\alpha \mid\!\sim^{\infty}_{LTM} \beta \quad \textit{if and only if} \quad \alpha \mid\!\sim^{mc}_{LTM} \beta$$

Proof. Due to Definition 7, the proof is based on the definition of $WM(LTM, \alpha, m)$ which returns a consistent sub-base WM such that by construction all its clauses belong to the same cluster of the LTM (since in WM_1 the clauses have at least one common variables with α, then WM_2 is a set of clauses that have at least one common variables with WM_1 and so on). Moreover m being big enough to contain any cluster of the LTM, downloading is limited to at most mc formulas, hence $WM(LTM, \alpha, \infty) = WM(LTM, \alpha, mc)$. □

The following propositions guarantees that the rejection of a query by using Working Memory is in accordance with the result that could be obtained by selecting a maximal consistent subbase of the LTM.

Proposition 2. *For all $m > 0$, if $\alpha \mid\!\sim^{m}_{LTM} \beta$ then there is a maximal consistent subbase B of the LTM s.t. $B \vdash \alpha \rightarrow \beta$*

Proof. By construction, for any m, $WM(LTM, \varphi, m)$ is a consistent subbase, thus there is maxi-consistent subbase of the LTM that contains it. Hence the result due to the monotonicity of $\vdash$. □

In the following, we compute the worst case time complexity denoted $T_{\max}$ associated to the processing of k consecutive queries, where processing a query means to check its consistency w.r.t. the current WM. The complexity of this processing is expressed w.r.t. the number of variables n and the number of clauses in the LTM denoted m_L, the capacity of WM in clauses is denoted m and the number of queries k of m_q clauses. When the process is done on the LTM only, the complexity is denoted $T_{\max}(LTM(n, m_L, k, m_q))$ while the one of the process done with a WM given the LTM is denoted $T_{\max}(\mathrm{WM}(n, m_L, m, k, m_q))$.

As recalled in [34] there is a sequence of papers that have provided algorithms for CNF_SAT with $2^{n-o(n)}.poly(m)$ runtime, where n is the number of variables and m is the number of clauses and $poly(m)$ is a polynomial function of m. The current best [11] is a deterministic algorithm that runs in $2^{n(1-\frac{1}{O(\log(m/n))})}poly(m)$ time, as shown by [16]. Applying these results in our context, the following remark shows the worst case computational complexity of checking k queries of m_q clauses in the LTM (please note that the expression is simpler when we assume that the number of queries and their size is negligible in front of the size of the LTM).

Remark 1. *If $k.m_q \ll m$ then $T_{\max}(LTM(n, m_L, k, m_q)) \in \Theta(k \times poly(m_L) \times 2^{n\alpha(n,m_L)})$ where $\alpha(n, m_L) = 1 - \frac{1}{O(\log(m_L/n))}$.*

This is due to the fact that $T_{\max}(LTM(n, m_L, k, m_q)) = \sum_{i=1}^{k}(T_{\max}(SAT(n, m_L + i.m_q)))$.

Due to Algorithms 1, 2 and 3, assessing k queries of m_q clauses via the WM has the following worst case computational complexity.

Proposition 3. *If $m_q \ll m$ then $T_{\max}(WM(n, m_L, m, k, m_q)) \in \Theta(n.m_L^2 + k.m_L.m^2 + k.poly(m) \times 2^{n.\alpha(n,m)})$*

Function QRandGener(LTM,$maxV$,$maxCl$)

Require: *RandNum(M): returns number in interval* $[1, M]$ *and Sample(S,n): returns n random elements from S and Vars(S): returns the variables of clauses set S*

Input: LTM: set of formulas in CNF format; $maxV$: maximum size of a query clause; $maxCl$: maximum number of clauses in query

Output: φ: query in CNF

```
counterCl ← 0; φ ← empty list of clauses
for counterCl in [1,RandNum(maxCl)] do
    clause ← empty list of literals
    vars ← Sample(Vars(LTM),RandNum(maxV))
    for each v in vars do
        newLit ← RandChoice({v̄, v}); clause ← newLit :: clause
    φ ← clause :: φ
return (φ)
```

Proof. $T_{\max}(WM(n, m_L, m, k, m_q)) =$

$$T_{\max}(LTMprep(n, m_L)) + \quad k \times \begin{pmatrix} T_{\max}(\text{Qprep}(n, m_L, m_q)) + \\ T_{\max}(WMdl(n, m_L, m, m_q)) + \\ m \times (T_{\max}(pop + push)) \\ + T_{\max}(SAT(n, m + m_q)) \end{pmatrix}$$

where push and pop are the operations that respectively add and delete an element from a fifo (here they are used to delete and add clauses to the *WM* since in the worse case m clauses have to replace all the clauses that were in the *WM* before, these operations can be implemented in $\Theta(1)$), *LTMprep*, Qprep, *WMdl* are the respective abbreviations for the Algorithms *LTM*_preprocessing (Algo 1), Query_preprocessing (Algo 2) and *WM*_download (Algo 3).
Moreover $T_{max}(LTMprep(n, m_L)) \in \Theta(n.m_L^2)$, since it computes the AssocCl dictionnary of the m_L clauses. $T_{max}(Qprep(n, m_L, m_q)) \in \Theta(m_L.m_q.n)$ considering that the intersection of two clauses is done in linear time of the number of variables (the clause literals being ordered) and this intersection being done between all the m_L clauses of the *LTM* and all the m_q clauses of the query. $T_{max}(WMdl(n, m_L, m, m_q)) \in \Theta(m_q.m_L + m.(m + m.m_L + m\log(m)))$ (since the while is done at worst m times and runs one membership test (in $\Theta(m)$), one inclusion test to the MUS list (in $\Theta(m.m_L)$ assuming that the clauses in the MUSes are ordered), one `Remove` and at worst m `Add`s to a heap of capacity size m which are both in $\Theta(\log m)$). After simplification, we get $T_{max}(WMdl(n,\ m_L,\ m,\ m_q) \in \Theta(m^2.m_L)$. Wrapping it up yields $T_{\max}(WM(n, m_L, m, k, m_q)) \in \Theta(n.m_L^2 + k(m_L.m_q.n + m^2.m_L + poly(m + m_q) \times 2^n)))$, we finally obtain the result. □

Hence if we ignore the first preprocessing of the *LTM*, comparing Rem. 1 and Prop. 3 leads to a theoretical gain in time in the worst case provided that $m \ll m_L$, *i.e.*, when the size of the *WM* is small w.r.t. the size of the LTM. This is confirmed by the following empirical results.

4.2 Empirical results

In order to assess the empirical interest of our approach, we implemented[5] the approach and placed ourselves in the situations where the base is receiving several consecutive queries. These queries are randomly generated from the clauses in the base (see Function QRandGener). Intuitively, the function creates a random number of clauses (bounded by a parameter $maxCl$) that are populated by a random number of variables (bounded by a parameter $maxV$) that appear in some clauses of the base (or a specific cluster); each of these variables is then randomly set positive or negative. Please note that it is possible to generate queries on a specific cluster of the base by replacing Vars(*LTM*) line 5 by Vars(*Cluster*); clusters are computed by transitive closure of the neighborhood relation between clauses (where neighbor means to have common variables, see Definition 6).

[5]Using Python 3.9.2, Sat4J 2.3.5 and CAMUS 1.0.7.

Query type	Filename	*WM MaxiCons* (Tbuild, Tsat)	*WM LTMSize* (Tbuild, Tsat)	*WM MaxCluster* (Tbuild,Tsat)	*WM AverageCluster* (Tbuild,Tsat)
q11	adi60 (1st)	(1299.02,17.8)	(3.3,6.1)	100% (2.43,6.1)	100% (2.32,6.1)
	adi60 (5th)	(2.03,14.3)	(1.72,5.8)	100% (1.16,5.9)	100% (1.22,5.8)
	mdi20 (1st)	(1299.02,17.8)	(7.84,7.2)	100% (8.68,7.1)	100% (8.25,7.3)
	mdi20 (5th)	(1.51,17.5)	(1.32,7.2)	97% (1.36,7.1)	97% (1.41,7.1)
	uma6 (1st)	(1181.35,16.4)	(4.02,6.6)	100% (5.03,6.6)	100% (3.03,6.5)
	uma6 (5th)	(1.29,14.9)	(1.08,6.5)	100% (1.33,6.4)	97% (1.51,6.3)
q33 on same cluster	adi60 (1st)	(1034.53,15.8)	(3.54,6.0)	100% (2.85,6.0)	100% (2.53,6.0)
	adi60 (5th)	(2.72,15.5)	(1.6,5.8)	98% (1.36,5.9)	98% (1.25,5.9)
	mdi20 (1st)	(1304.62,18.2)	(8.08,7.2)	100% (7.86,7.3)	100% (8.44,7.2)
	mdi20 (5th)	(2.17,17.3)	(1.4,7.0)	99% (1.32,7.1)	99% (1.58,7.2)
	uma6 (1st)	(1166.63,16.6)	(6.14,6.7)	100% (5.15,6.6)	99% (2.81,6.5)
	uma6 (5th)	(2.37,16.1)	(1.31,6.2)	100% (1.25,6.2)	92% (1.58,6.3)
q33 on different clusters	adi60 (1st)	(1025.18,15.3)	(14.36,7.2)	83% (3.02,6.6)	83% (2.89,6.5)
	adi60 (5th)	(2.53,14.9)	(5.74,9.4)	85% (3.1,6.6)	85% (2.73,6.6)
	mdi20 (1st)	(1285.47,18.1)	(57.22,9.1)	89% (6.19,7.6)	89% (6.79,7.5)
	mdi20 (5th)	(2.1,17.5)	(13.74,11.8)	89% (5.56,7.8)	89% (5.86,7.6)
	uma6 (1st)	(1174.62,16.9)	(36.65,8.6)	90% (7.02,7.4)	86% (4.66,7.0)
	uma6 (5th)	(2.22,16.5)	(9.15,11.5)	92% (5.68,7.6)	94% (4.17,7.0)

Table 1: Agreement ratio between differently sized *WM* for 100 runs. Percentages in the cells correspond respectively to agreement ratio between the *WM LTMSize* and, respectively, *WM MaxCluster* and *WM AverageCluster*. Tbuild and Tsat represent respectively the time (in ms) to build the *WM* and to execute the sat solver.

In order to assess our approach, we observed its results under different maximal sizes for the WM. As a baseline, a WM representing the most relevant maxi-consistent subbase is computed (*WM MaxiCons*); this WM is created by associating the query with all the clauses from the base (assigning 0 in case no variables are shared) and by using the regular *WM_download* algorithm. We then run the experiment with *WM* having respectively the size of the *LTM* (*WM* LTMSize), the size of the cluster of clauses of maximum size (*WM* MaxCluster) and the size of the average size of all the clusters of clauses (*WM* AverageCluster).

Table 1 summarizes the results for different bases built on two Dimacs files coming from the SAT benchmark Blocks World[6]: *mdi* and *ami* are respectively Medium.cnf cut off after 150 lines and Anomaly.cnf cut off after 50 lines, both of them made inconsistent by negating their first clause; *mdiX* and *amiX* are the files obtained by repeating the *mdi* and *ami* X times (literals are renamed to avoid redundancy); *uma* is the union of *mdi2* and *ami4*. Hence, *adi60*, *mdi20* and *um6* are all composed of 3000 clauses.

The notation qXY indicates that the generated query has a maximum number of clauses of X and a maximum number of variables per clause of Y. (1st) and (5th) indicates respectively that the row corresponds to the first or the last queries of five successive queries.

Based on the results on the table, we can make the following remarks. Bases that have several connected components benefit from the approach when the query concerns a limited amount of these components, since the solver will be executed on a much smaller base which will reduce the execution time while maintaining accuracy. We argue however that it is fair to assume that a general base will have a tendency to cluster its formulas, where each cluster represents some sort of "context" or "domain".

Iterative querying on the same cluster allows to reduce of lot of the overhead caused by the approach since the *WM* does not need to be recomputed. On the other hand, iterative querying on the whole base forces the re-computation of the *WM* quite often, which implies longer execution times. In that case, the accuracy may decrease since the working memory may be overwhelmed

[6] https://www.cs.ubc.ca/~hoos/SATLIB/Benchmarks/SAT/PLANNING/BlocksWorld/descr.html

by the number of subjects that must be covered at the same time; it is interesting to note that this behavior is somewhat reminiscent of human behavior, for instance when a person, by mixing different subjects and domains, makes it impossible to apprehend the full extent of her statement.

Finally, as expected, let us note that the choice of a suitable *WM* capacity is a matter of compromise between time and correctness: a bigger size will give more correct results but will take more time to compute whereas a smaller size may be less correct but faster. That being said, as Table 1 shows, selecting a size equal to the average of the clusters demonstrates noticeable decrease in time while maintaining good results.

5 Conclusion

In this paper, we presented an approach to handle inconsistency in a knowledge base by using a notion of associations between clauses based on common variables. These associations are used to extract one consistent subbase. We showed that this approach has interesting results both in terms of complexity and execution time.

It should be noted that our approach, by eliciting only one subbase, may give results that are different from some classical approaches that consider all the consistent subbases. Moreover, depending on the history of the knowledge base, i.e. the sequence of queries that happened beforehand, different consistent subbases can be chosen. In addition, we can remark that our approach behaves in a different way than MAXSAT (which also selects only one subbase). This can be observed in the following small example:

Example 6. *Let us consider a KB LTM' in CNF form:*

$$LTM' = \begin{array}{l} \overbrace{(a \vee b \vee c)}^{c_1} \wedge \overbrace{\neg a \vee \neg d}^{c_2} \wedge \overbrace{(a \vee \neg c \vee \neg d)}^{c_3} \wedge \\ \overbrace{\neg b \vee c}^{c_4} \wedge \overbrace{(\neg d \vee \neg e)}^{c_5} \wedge \overbrace{a \vee e}^{c_6} \wedge \overbrace{c \vee e}^{c_7} \wedge \\ \overbrace{(\neg c \vee d)}^{c_8} \wedge \overbrace{(\neg a \vee c)}^{c_9} \wedge \overbrace{(\neg e)}^{c_{10}} \end{array}$$

Let $\varphi = (c \vee d) \wedge (\neg b)$. In order to minimize the number of deleted clauses, MAXSAT would find a solution by removing only the clause c_8, and φ would be accepted. On the contrary, our approach would create the WM by selecting all the clauses but c_9 and c_{10} and reject the query. In that example, it seems more desirable to exclude c_9 and c_{10} since they are less related to the query than c_8.

This first preliminary study opens several research avenues:

Finer WM building: extending the relevance between clauses by being able to compare them semantically instead of just counting their common variables (see e.g. [9] where associated formulas are built on the results of a serious game) could overcome the drawbacks related to the syntax dependency of the non-monotonic inference relation $\mid\!\sim^{\varphi}_{LTM}$. It is important to note that, with the current syntactic definition of relevance, different sets of clauses may be relevant to two equivalent clauses (e.g. by disjunctively adding superfluous literals): for instance, consider the clauses c and $a \vee \neg a \vee c$. On that note, the reader can check that $\mid\!\sim^{m}_{LTM}$ satisfies some classical properties of non-monotonic inference relations of [26] like reflexivity ($\alpha \mid\!\sim^{m}_{LTM} \alpha$) and right weakening (if $\vdash \alpha \rightarrow \beta$ and $\gamma \mid\!\sim^{m}_{LTM} \alpha$ then $\gamma \mid\!\sim^{m}_{LTM} \beta$). However, left logical equivalence, cut or cautious monotony[7] are not guaranteed since two equivalent formulas may imply different WM downloading. Other, more semantical, definitions of relevance between clauses may allow for the satisfaction of more non-monotonic properties, for instance the semantical dependence built on the notion of forgetting [28].

[7] The reader can refer to [27] for a well-organized overview of the main classical non-monotonic inference relations and their properties (in French) or to [13, 14] for its English counterparts.

WM & LTM updating: an interesting study would focus on the evolution of the knowledge with the arrival of different queries, i.e. under which conditions the formula of the query might be accepted and stored in the *WM*. Moreover, considering a capacity limited *WM* implies that some *WM* clauses might be discarded to make room for others clauses when a query is irrelevant to the current *WM*. These currently unnecessary clauses might still be relevant for later incoming queries and purely losing them might be detrimental in the long run. One perspective is hence to study in detail which clauses should be unloaded from *WM* and stored in the *LTM* and, in order to avoid too much redundancy, how those clauses could be compacted in the *LTM*. Updating the *LTM* prompts then the computation of a new association table to account for the new pieces of information, which may be done efficiently by using the old AssocCl together with QAssocV. In this context, an incremental algorithm has to be created in order to update the MUSes associated to the updated LTM.

Different inconsistency handling: Our approach handles inconsistency based on the idea that inconsistent pieces of information lead to concealing some other formulas, meaning that depending on the context some knowledge will be ignored. Introducing uncertainty on the formula, for instance with penalty logic [23], would ensure that every piece of information is taken into consideration, albeit with different "strength".

Acknowledgment

The paper has benefited from pertinent remarks made by anonymous reviewers of ECSQARU'2021, and also from very interesting points raised by anonymous reviewers of NMR'2021. We thank them all for their very useful work.

References

[1] Leila Amgoud, Philippe Besnard, Claudette Cayrol, Philippe Chatalic, and Marie-Christine Lagasquie-Schiex. Argumentation and inconsistency-tolerant reasoning. In Pierre Marquis, Odile Papini, and Henri Prade, editors, *A Guided Tour of Artificial Intelligence Research: Volume I: Knowledge Representation, Reasoning and Learning*, pages 415–440. Springer Nature, 2020.

[2] Gilles Audemard, Jean-Marie Lagniez, and Laurent Simon. Improving glucose for incremental sat solving with assumptions: Application to MUS extraction. In *International conference on theory and applications of satisfiability testing*, pages 309–317. Springer, 2013.

[3] Arnon Avron, Ofer Arieli, and Anna Zamansky. *Theory of effective propositional paraconsistent logics*. College Publications, 2018.

[4] Alan D Baddeley and Graham Hitch. Working memory. In *Psychology of learning and motivation*, volume 8, pages 47–89. Elsevier, 1974.

[5] Pierre Barrouillet and Valérie Camos. The time-based resource-sharing model of working memory. *Working memory: State of the science*, page 85, 2007.

[6] Salem Benferhat, Claudette Cayrol, Didier Dubois, Jérôme Lang, and Henri Prade. Inconsistency management and prioritized syntax-based entailment. In *IJCAI*, volume 93, pages 640–645, 1993.

[7] Salem Benferhat, Didier Dubois, and Henri Prade. How to infer from inconsisent beliefs without revising? In *IJCAI*, volume 95, pages 1449–1455. Citeseer, 1995.

[8] Leopoldo Bertossi. Consistent query answering in databases. *ACM Sigmod Record*, 35(2):68–76, 2006.

[9] Pierre Bisquert, Madalina Croitoru, Florence Dupin de Saint-Cyr, and Abdelraouf Hecham. Formalizing cognitive acceptance of arguments: Durum wheat selection interdisciplinary study. *Minds and Machine*, 27(1):233–252, 2017.

[10] Gerhard Brewka. Preferred subtheories: An extended logical framework for default reasoning. In *IJCAI*, volume 89, pages 1043–1048. Citeseer, 1989.

[11] Chris Calabro, Russell Impagliazzo, and Ramamohan Paturi. A duality between clause width and clause density for sat. In *21st Annual IEEE Conference on Computational Complexity (CCC'06)*, pages 7–pp. IEEE, 2006.

[12] Walter Alexandre Carnielli and Marcelo Esteban Coniglio. *Paraconsistent logic: Consistency, contradiction and negation*, volume 40. Springer, 2016.

[13] Claudette Cayrol and Marie-Christine Lagasquie-Schiex. Non-monotonic syntax-based entailment: A classification of consequence relations. In Christine Froidevaux and Jürg Kohlas, editors, *Symbolic and Quantitative Approaches to Reasoning and Uncertainty, European Conference, ECSQARU'95, Fribourg, Switzerland, July 3-5, 1995, Proceedings*, volume 946 of *Lecture Notes in Computer Science*, pages 107–114. Springer, 1995.

[14] Claudette Cayrol, Marie-Christine Lagasquie-Schiex, and Thomas Schiex. Nonmonotonic reasoning: From complexity to algorithms. *Ann. Math. Artif. Intell.*, 22(3-4):207–236, 1998.

[15] Jan Chomicki. Consistent query answering: Opportunities and limitations. In *17th International Workshop on Database and Expert Systems Applications (DEXA'06)*, pages 527–531. IEEE, 2006.

[16] Evgeny Dantsin and Edward A Hirsch. Worst-case upper bounds. *Handbook of Satisfiability*, 185(403-424):9, 2009.

[17] Wim De Neys and Deborah Everaerts. Developmental trends in everyday conditional reasoning: The retrieval and inhibition interplay. *Journal of Experimental Child Psychology*, 100(4):252–263, 2008.

[18] Wim De Neys, Walter Schaeken, and Géry d'Ydewalle. Working memory and everyday conditional reasoning: Retrieval and inhibition of stored counterexamples. *Thinking & Reasoning*, 11(4):349–381, 2005.

[19] Dragan Doder and Srdjan Vesic. How to decrease and resolve inconsistency of a knowledge base?. In *ICAART (2)*, pages 27–37, 2015.

[20] Didier Dubois, Patricia Everaere, Sébastien Konieczny, and Odile Papini. Main issues in belief revision, belief merging and information fusion. In Pierre Marquis, Odile Papini, and Henri Prade, editors, *A Guided Tour of Artificial Intelligence Research: Volume I: Knowledge Representation, Reasoning and Learning*, pages 441–485. Springer Nature, 2020.

[21] Didier Dubois and Henri Prade. Possibilistic logic - an overview. In Jörg H. Siekmann, editor, *Computational Logic*, volume 9 of *Handbook of the History of Logic*, pages 283–342. Elsevier, 2014.

[22] Florence Dupin De Saint Cyr, Béatrice Duval, and Stéphane Loiseau. A priori revision. In Salem Benferhat and Philippe Besnard, editors, *European Conference on Symbolic and Quantitative Approaches to Reasoning with Uncertainty (ECSQARU 2001), Toulouse, 19/09/01-21/09/01*, number 2143 in LNAI, pages 488–497, http://www.springerlink.com, 2001. Springer.

[23] Florence Dupin de Saint-Cyr, Jérôme Lang, and Thomas Schiex. Penalty logic and its link with dempster-shafer theory. In *Uncertainty Proceedings 1994*, pages 204–211. Elsevier, 1994.

[24] Sergio Greco, Cristina Sirangelo, Irina Trubitsyna, and Ester Zumpano. Preferred repairs for inconsistent databases. In *Seventh International Database Engineering and Applications Symposium, 2003. Proceedings.*, pages 202–211. IEEE, 2003.

[25] Matti Järvisalo, Daniel Le Berre, Olivier Roussel, and Laurent Simon. The international sat solver competitions. *Ai Magazine*, 33(1):89–92, 2012.

[26] Sarit Kraus, Daniel Lehmann, and Menachem Magidor. Nonmonotonic reasoning, preferential models and cumulative logics. *Artificial intelligence*, 44(1-2):167–207, 1990.

[27] Marie-Christine Lagasquie-Schiex. *Contribution à l'étude des relations d'inférence non-monotone combinant inférence classique et préférences. (A Contribution to the study of non-monotonic inference relationships combining classical inference and preferences)*. PhD thesis, Paul Sabatier University, Toulouse, France, 1995.

[28] Jérôme Lang, Paolo Liberatore, and Pierre Marquis. Propositional independence: Formula-variable independence and forgetting. *J. Artif. Intell. Res.*, 18:391–443, 2003.

[29] Daniel Le Berre and Anne Parrain. The sat4j library, release 2.2. *Journal on Satisfiability, Boolean Modeling and Computation*, 7(2-3):59–64, 2010.

[30] Chu Min Li and Felip Manya. Maxsat, hard and soft constraints. In *Handbook of satisfiability*, pages 613–631. IOS Press, 2009.

[31] Mark H Liffiton and Karem A Sakallah. Algorithms for computing minimal unsatisfiable subsets of constraints. *Journal of Automated Reasoning*, 40(1):1–33, 2008.

[32] John McCarthy. Applications of circumscription to formalizing common-sense knowledge. *Artificial intelligence*, 28(1):89–116, 1986.

[33] Bernhard Nebel. Belief revision and default reasoning: Syntax-based approaches. *KR*, 91:417–428, 1991.

[34] Mihai Pătraşcu and Ryan Williams. On the possibility of faster sat algorithms. In *Proceedings of the twenty-first annual ACM-SIAM symposium on Discrete Algorithms*, pages 1065–1075. SIAM, 2010.

[35] Judea Pearl. System z: A natural ordering of defaults with tractable applications to nonmonotonic reasoning. In *Proceedings of the 3rd Conference on Theoretical Aspects of Reasoning about Knowledge*, pages 121–135, 1990.

[36] N. Rescher and R. Manor. On inference from inconsistent premises. *Theory and Decision*, 1:179–219, 1970.

[37] Herbert A Simon. A behavioral model of rational choice. *The quarterly journal of economics*, 69(1):99–118, 1955.